Rapport Projet : Métro c'est trop

Anis BOUZIANE Julien JACQUET Damien DEMONTIS

5 mai 2017

Note : La methode utilisée pour écrire le code ne permet pas d'afficher les accents et les lignes trop longues sont coupées, cependant l'intégralité du code est également disponible à l'adresse suivante : https ://github.com/SolowinFXVI/Metrocesttrop

1 Explication des structures de données :

La premiere structure $SOMMET$ sert à stocker toutes les informations relative à un sommet/station (index du sommet, nom de la station, ligne de métro, terminus ou non). Pour stocker ces informations on va utiliser des tableau de *char* avec des tailles prédéfinies, cette technique consomme plus de mémoire que nécéssaire mais évite l'utilisation de *mallocs*.

La seconde structure TAB stock les informations relatives à tous les sommets/stations. C'est un tableau de la structure précédente.

La troisième structure ARC contient les informations relatives a l'existence d'un arc c'est a dire un sommet de depart de type $SOMMET$, un sommet d'arrivée de type $SOMMET$ également, ainsi que le poids de cet arc ici défini par le temps de trajet entre ces deux sommets.

La quatrième structure G est la matrice d'adjascence du graph elle est de type ARC et contient par conséquent toutes les informations des structures précédentes. On retrouve en abscisse et en ordonnées tous les sommets existants. Puisque notre graph est non orienté il faut faire en sorte que les arcs aillent dans les deux sens.

2 Principales procédures :

La principale procédure est la fonction "plus court chemin" sans aucun doute mais c'est au coeur de cette fonction que se trouve la partie la plus importante du programme : la partie $dijkstra$. Pour résoudre notre problème de plus court chemin nous avons décidé d'implémenter l'algorithme de $dijkstra$. Tout d'abord parce que c'est celui que nous avons vu en cours mais aussi parce que c'est un algorithme réputé. Il existe d'autres solutions au probleme du plus court chemin mais dijsktra semblait l'algorithme le plus adapté.

Il serait intéréssant de savoir quel algorithme est utilisé par l'application de la sncf/ratp.Est-ce l'algorithme de dijkstra ? Gourmand mais qui donne dans tous les cas le plus court chemin. Ou un algorithme de type A*, moins précis mais plus rapide ?

Une grosse partie du projet est basé sur l'utilisation des *string* nous avons décidés de les utiliser tout le long pour simplifier les algorithmes cependant cela entraine l'appartion de très nombreuses petites conversions, d'où la grande quantitée de fonction *atoi* (conversion d'un caractère ascii en int ex :"99999"->99999) et de *strcpy* qui est une façon de remplir un tableau avec des string facilement (nous avions des problèmes avec le remplissage et cette fonction nous à grandement simplifiée la vie).

Il est parfois difficile de distinguer a quel moment utiliser les valeurs "NBR STATIONS" et "NBR ARCS" et cela peut porter a confusion mais le projet fonctionne et il n'y a pas d'érreurs de mémoire.

Déroulement simplifié du programme On commence avec les routines d'arguments, puis on va initialiser les sommets d'abord avec du "vide" ("UNKNOWN") pour nous aider en cas d'erreurs (si on peut lire unknown mais pas les données on sait d'où vient le problème). Ensuite on extrait les données relatives aux sommets depuis metro.txt. On initilise ensuite la structure G, le graph, là encore on utilise la meme technique du remplissage avec du "vide", puis on remplie avec les données. Enfin on arrive a plus court chemin, où on va appliquer dijkstra et où on va se servir d'une liste pour stocker les étapes du trajet. Quand dijkstra est terminé on défait la liste et obtient le trajet complet. Une petite conversion du temps en minutes et voilà ! Tout au long du programme on peut apercevoir des *printf* indiquants quelle fonction est utilisée elles permettent de savoir ce que le programme est en train de faire et ne gennent pas l'affichage du résultat.

Pour mener à bien ce projet il à fallut modifier grandement le fichier *metro.txt*, de façon à obtenir un formatage pratique à utiliser. Nous avons eu des difficultés avec la partie acquisition des donées et finalement le programme est fonctionnel mais sans affichage graphique.

Améliorations possibles Pour améliorer notre programme on pourrait y ajouter une touche graphique avec peut etre les coordonnées géographiques des stations sur une carte de Paris, mais le coeur du programme resterait inchangé. On pourrait également avoir une base de données plus complète et pratique, malgré de nombreux changement du fichier *metro.txt* (plus de 1000 operations manuelles) il reste incomplet et peut pratique en raison notamment des *string*,des apostrophes et des espaces(tous ces problèmes ont étés résolus en mettant des underscores à chacun d'eux). On pourrait égelement ajouter au fichier *metro.txt* les directions ce qui faciliterait la lisibilité pour l'utilisateur (meme si pour l'instant l'utilisateur a juste besoin de lire sur les panneaux dans la station jusqu'où il doit aller pour comprendre la direction).

3 CODE COMPLET :

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "lecture.h"
#include "mct.h"
#include "dijkstra.h"

TAB initialise_sommets(char *str, TAB M){ //fait appel aux fonctions de lecture.c pour initialiser les sommets
  printf("initialise_sommets\n");
  M=init_s(M);
  return initialiser_sommets(str, M);
}

int conversion_sommet_str_sommet_int(TAB M, char *sommet_str){ //va chercher la valeur d'index du sommet entr  en "string"
  printf("conversion_sommet_str_sommet_int\n");
  int sommet=-1;
  int i;

  for(i=0; i<NBR_STATIONS;i++){
    if(strcmp(sommet_str,M.TAB[i].nom)==0){
      sommet = atoi(M.TAB[i].index);
      return sommet;
    }
  }
  if(sommet == -1){
    printf("sommet inconnu ou mal orthographi \n");
    printf("ex1: Basilique de Saint-Denis -> Basilique_de_Saint-Denis\n");
    printf("ex2: Place d'Italie -> Place_d_Italie\n");
    exit(EXIT_FAILURE);
  }
  return sommet;
}

////////////////////////////////////////////////////////////////////////////////
int main(int argc, char *argv[]) {
  TAB M;
  char *sommet_depart_str;
  char *sommet_arrivee_str;

  if(argc < 3){ //routines de lectures d'arguments
    printf("\n");
    printf("Trop peut d'arguments\n");
    printf("utilisation :\n");
    printf("./mct station_de_depart station_d'arrivee\n");
    printf("\n");
    exit(EXIT_FAILURE);
  }
```

```
48
49    if(argc == 3){//stockage des sommets entr s par
          l'utilisateur sous forme de "string"
50      sommet_depart_str=argv[1];
51      sommet_arrivee_str=argv[2];
52    }
53
54    if(argc>3){//routines d'arguments
55      printf("Trop d'arguments\n");
56      printf("utilisation :\n");
57      printf("mct station_de_depart station_d'arrivee\n");
58      exit(EXIT_FAILURE);
59    }
60
61    M=initialise_sommets("metro.txt",M);
62    printf("sommets M initialis s\n");
63    initialise_graph("metro.txt",G,M);
64    printf("graph initialis \n");
65    plus_court_chemin(G,M,conversion_sommet_str_sommet_int(M,sommet_depart_str),conve
66    printf("fin algorithme\n");
67    return 0;
68  }
69  ////////////////////////////////////////////////////////////////////////////
70  //DEMONTIS BOUZIANE JACQUET
```

../src/mct.c

```c
#ifndef __MCT_H
#define  __MCT_H

#include "constantes.h"
#include "structures.h"
#include "lecture.h"
//STAT initialiser_sommets_et_arcs(char *str);

void calcul_trajet();

void afficher_trajet();

#endif
//DEMONTIS BOUZIANE JACQUET
```

../src/mct.h

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "lecture.h"

TAB init_s(TAB M){ //remplie le tableau des sommet d'un marqueur "UNKNOWN"
  int i;
  for(i=0; i<NBR_STATIONS-1; i++){
    strcpy(M.TAB[i].index,"UNKNOWN");
    strcpy(M.TAB[i].nom,"UNKNOWN");
    strcpy(M.TAB[i].ligne,"UNKNOWN");
    strcpy(M.TAB[i].status,"UNKNOWN");
  }
  return M;
}

void init_g(ARC G[NBR_ARCS][NBR_ARCS]){//remplie la structure G de marqueurs "UNKNOWN"
  int i;
  int j;
  for(i=0;i<NBR_ARCS;i++){
    for(j=0;j<NBR_ARCS;j++){
      strcpy(G[i][j].sm1.index,"UNKNOWN");
      strcpy(G[i][j].sm1.nom,"UNKNOWN");
      strcpy(G[i][j].sm1.ligne,"UNKNOWN");
      strcpy(G[i][j].sm1.status,"UNKNOWN");
      strcpy(G[i][j].sm2.index,"UNKNOWN");
      strcpy(G[i][j].sm2.nom,"UNKNOWN");
      strcpy(G[i][j].sm2.ligne,"UNKNOWN");
      strcpy(G[i][j].sm2.status,"UNKNOWN");
      strcpy(G[i][j].temps,"99999");

    }
  }
}

TAB initialiser_sommets(char *str, TAB M){ //extrait les donn es de "metro.txt" et les stock dans le tableau des sommets
  int i;
  char type[128];
  char index[128];
  char nom[128];
  char ligne[128];
  char status[128];
  FILE* fic = fopen(str, "r");

  if (fic == NULL) {
    printf("echec ouverture fichier%s\n", str);
    exit(EXIT_FAILURE);
  }

  for(i=0;i<NBR_STATIONS;i++){ //ignore ce qu'il y a apr s
```

```
      la derniere station
51    fscanf(fic,"%s %s %s %s %s\n", &type[0], &index[0],
          &nom[0], &ligne[0], &status[0]);
52    strcpy(M.TAB[i].index,index);
53    strcpy(M.TAB[i].nom,nom);
54    strcpy(M.TAB[i].ligne,ligne);
55    strcpy(M.TAB[i].status,status);
56  }
57
58 fclose(fic);
59 return M;
60 }
61
62 void associer_graph_data(char *sm1, char *sm2,ARC
      G[NBR_ARCS][NBR_ARCS],TAB M){
63   //associe les donn es stock es dans le tableau des
        sommets a la structure G
64   int i;
65   for(i=0;i<NBR_STATIONS;i++){
66     if(atoi(sm1) == atoi(M.TAB[i].index)){
67       strcpy(G[atoi(sm1)][atoi(sm2)].sm1.index,M.TAB[i].index);
68       strcpy(G[atoi(sm1)][atoi(sm2)].sm1.nom,M.TAB[i].nom);
69       strcpy(G[atoi(sm1)][atoi(sm2)].sm1.ligne,M.TAB[i].ligne);
70       strcpy(G[atoi(sm1)][atoi(sm2)].sm1.status,M.TAB[i].status);
71     }
72     if(atoi(sm2) == atoi(M.TAB[i].index)){
73       strcpy(G[atoi(sm1)][atoi(sm2)].sm2.index,M.TAB[i].index);
74       strcpy(G[atoi(sm1)][atoi(sm2)].sm2.nom,M.TAB[i].nom);
75       strcpy(G[atoi(sm1)][atoi(sm2)].sm2.ligne,M.TAB[i].ligne);
76       strcpy(G[atoi(sm1)][atoi(sm2)].sm2.status,M.TAB[i].status);
77     }
78   }
79 }
80
81 void initialiser_graph(char *str,ARC G[NBR_ARCS][NBR_ARCS],
      TAB M){ //extrait les donn es de "metro.txt" et les
      stock dans le tableau des arcs
82   int i=0;
83   char type[128];
84   char sm1[128];
85   char sm2[128];
86   char temps[128];
87   char garbage[128];
88   FILE* fic = fopen(str, "r");
89
90   if (fic == NULL) {
91     printf("echec ouverture fichier%s\n", str);
92     exit(EXIT_FAILURE);
93   }
94
95   while(fscanf(fic,"%s %s %s %s %s\n", &type[0], &sm1[0],
        &sm2[0], &temps[0], &garbage[0]) != EOF){
96       if(i>=376){ //ignore toute la partie sur les sommets
97         associer_graph_data(sm1,sm2,G,M);
```

```
98          associer_graph_data(sm2,sm1,G,M);
99          strcpy(G[atoi(sm1)][atoi(sm2)].temps,temps);
100         strcpy(G[atoi(sm2)][atoi(sm1)].temps,temps);
101         i++;
102       }
103     i++;
104   }
105 fclose(fic);
106 }
107
108 void initialise_graph(char *str,ARC
      G[NBR_ARCS][NBR_ARCS],TAB M){ //fonction
      d'initialisation complete de la structure G
109   printf("initialise_graph\n");
110   init_g(G);
111   initialiser_graph(str,G,M);
112 }
113 //DEMONTIS BOUZIANE JACQUET
```

../src/lecture.c

```c
#ifndef __LECTURE_H
#define __LECTURE_H

#include "constantes.h"
#include "structures.h"

TAB init_s(TAB M);
TAB initialiser_sommets(char *str, TAB M);
TAB cleanup(TAB M);

void initialise_graph(char *str, ARC G[NBR_ARCS][NBR_ARCS],
    TAB M);
void init_g(ARC G[NBR_ARCS][NBR_ARCS]);

#endif
//DEMONTIS BOUZIANE JACQUET
```

../src/lecture.h

```c
#include <stdio.h>
#include <stdlib.h>
#include "constantes.h"
#include "structures.h"
#include "string.h"

List fill_start(List l, int a){ //implementation classique d'une liste
  printf("fill_start\n");
  Element* new= malloc(sizeof(Element));
  new->val = a;
  new->next=l;
  l=new;
  return l;
}

void print_list_path(List l, TAB M,ARC G[NBR_ARCS][NBR_ARCS],int temps_total){//affiche la liste
  if(l==NULL || l->next == NULL){
    printf("liste vide\n");
    return ;
  }
    while(l->next != NULL){
      printf("De  %s  allez jusqu'a  %s ligne %s\n",M.TAB[l->val].nom, M.TAB[l->next->val].nom, G[l->val][l->next->val].sm2.ligne);
      l=l->next;
    }
}

int conversion_temps_min(int temps_total){ //simple conversion du temps en minutes
  int minutes;
  minutes=temps_total/60;
  return minutes;
}

int conversion_temps_sec(int temps_total){//va de paire avec la fonction conversion_temps_min et renvoie les secondes restantes
  int secondes;
  secondes=temps_total%60;
  return secondes;
}



int sommets_tous_traites(int sommets_traites[NBR_STATIONS]){//test si le tableau ne contient que des 1 -> retourne 1
  int i;
  for(i=0; i<NBR_STATIONS; i++){
    if(sommets_traites[i]==0){
      return 0;
```

```
46      }
47    }
48    return 1;
49  }
50
51  void init_pere(int pere[NBR_ARCS]){//initialise le tableau
        pere a -1
52    printf("init_pere \n");
53    int i;
54    for(i=0; i<NBR_STATIONS; i++){
55      pere[i]=-1;
56    }
57  }
58
59
60  void dijkstra(int pere[NBR_STATIONS], int
        sommet_depart){//Transforme le tabelau pere en tableau
        des plus courts chemins
61    printf("dijkstra \n");
62
63    int sommets_traites[NBR_STATIONS]={0};
64    int plus_courte_distance[NBR_STATIONS];
65    sommets_traites[sommet_depart] = 1;
66    int i;
67
68    init_pere(pere);
69
70    for(i=0; i < NBR_STATIONS; i++){ //initialisation du
          tableau plus courte distance
71      if(atoi(G[sommet_depart][i].temps) != INFINI){//si i
            successeur du sommet de depart
72        plus_courte_distance[i]=atoi(G[sommet_depart][i].temps);//alors
              plus courte distance vaut le poids de l'arc
73        pere[i]= sommet_depart;
74      }
75      else{ //sinon plus courte distance vaut l'infini
76        plus_courte_distance[i]=INFINI;
77      }
78    }
79
80    plus_courte_distance[sommet_depart] = 0;
81
82  ////////////////////////////////////////////////////////////////////////////
83    while(!sommets_tous_traites(sommets_traites)){
84      printf("tours de while\n");
85      //recherche du prochain sommet
86      int min = INFINI, a_traiter;
87
88      for(i=0; i<NBR_STATIONS; i++){//pour toutes les stations
89
90        if(sommets_traites[i] == 0){//si le sommet n'est pas
              trait
91          if(plus_courte_distance[i] <= min){//si la plus
                courte distance est inferieure au minimum actuel
```

```
92            a_traiter = i;
93            min=plus_courte_distance[i];
94          }
95        }
96      }
97
98      sommets_traites[a_traiter] = 1;
99
100     for(i=0; i<NBR_STATIONS;i++){//pour les successeurs du
            sommet a_traiter
101       if(atoi(G[a_traiter][i].temps) != INFINI){//test si
              la distance de ce sommet au sommet de depart est
              plus grande que celle passant par le sommet
              a_traiter
102         if(plus_courte_distance[i] >=
                plus_courte_distance[a_traiter] +
                atoi(G[a_traiter][i].temps)){
103           //mise a jour de la distance
104           plus_courte_distance[i] =
                  plus_courte_distance[a_traiter] +
                  atoi(G[a_traiter][i].temps);
105           //donne le pere
106           pere[i] = a_traiter;
107         }
108       }
109     }
110   }
111 ////////////////////////////////////////////////////////////////////////////
112 }
113
114 void plus_court_chemin(ARC G[NBR_ARCS][NBR_ARCS],TAB M, int
        sommet_depart, int sommet_arrivee){//calcul le plus
        court chemin entre deux sommets
115   printf(" plus_court_chemin\n");
116   if(sommet_depart == sommet_arrivee){
117     printf("Vous etes deja arriv ...");
118     return ;
119   }
120
121   int pere[NBR_ARCS]; //tableau utilis  par dijkstra
122   List path = NULL;//liste de stockage du plus court chemin
123   int temps_total = 0;
124
125   dijkstra(pere,sommet_depart); //COEUR DU PROGRAMME :
          modifie le tableau pere
126
127   int depart, arrivee = sommet_arrivee;
128
129   path = fill_start(path,arrivee);//remplissage du debut de
          la liste
130
131   do {
132     depart = pere[arrivee]; //sommet de depart actuel est
            le pere du sommet d'arriv e
```

```
133     if(depart == -1){
134       printf("Pas de chemin possible entre %s et
              %s\n",M.TAB[sommet_depart].nom,
              M.TAB[sommet_arrivee].nom);
135       return ;
136     }
137     temps_total += atoi(G[depart][arrivee].temps);
            //incr mentation du temps
138     path=fill_start(path,depart);//remplissage de la liste
139     arrivee=depart;
140   }while(depart != sommet_depart);
141   printf("\n");
142   print_list_path(path,M,G,temps_total);//affichage du
        trajet
143   printf("\n");
144   printf("temps de trajet : %d minutes et %d
        secondes\n",conversion_temps_min(temps_total),conversion_temps_sec(temps_total
145   //affichage du temps de trajet
146   printf("\n");
147 }
148 //DEMONTIS BOUZIANE JACQUET
```

../src/dijkstra.c

```c
#ifndef __DIJKSTRA_H
#define __DIJKSTRA_H

#include "structures.h"
void print_list_path(List path,TAB M,ARC
    G[NBR_ARCS][NBR_ARCS],int temps_total);
void dijkstra(int pere[NBR_STATIONS], int sommet_depart);

void plus_court_chemin(ARC G[NBR_ARCS][NBR_ARCS],TAB M, int
    sommet_depart, int sommet_arrivee);

#endif
//DEMONTIS BOUZIANE JACQUET
```

../src/dijkstra.h

```c
#ifndef __MCT_H
#define  __MCT_H

struct sommet{ //structure contenant les informations relatives a un sommet
  char index[128];
  char nom[128];
  char ligne[128];
  char status[128];
};
typedef struct sommet SOMMET;

struct tab{//structure contenant les informations relatives a tous les sommets
  struct sommet TAB[377];
};
typedef struct tab TAB;

struct arc{//structure contenant les informations relatives a l'existance d'un arc
  struct sommet sm1;
  struct sommet sm2;
  char temps[128];
};
typedef struct arc ARC;

ARC G[472][472]; //Matrice d'adjascence utilis e par la partie dijkstra

struct element{ //liste pour stocker le trajet
  int val;
  struct element* next;
};

typedef struct element Element;
typedef Element* List;

#endif
//DEMONTIS BOUZIANE JACQUET
```

../src/structures.h

```c
#define NBR_STATIONS 376
#define NBR_ARCS 472
#define TAILLE_NOM 200
#define TAILLE_LIGNE 5
#define INFINI 99999

//DEMONTIS BOUZIANE JACQUET
```

../src/constantes.h

```
CFLAGS=-c -g  -Wall

all:  mct.o lecture.o dijkstra.o
  make  clean
  make  mct.o
  make  lecture.o
  make  dijkstra.o
  gcc -o  mct mct.o lecture.o dijkstra.o
  ./mct

compil: mct
  make  mct

test: mct.o lecture.o dijkstra.o
  make  clean
  make  mct.o
  make  lecture.o
  make  dijkstra.o
  gcc -o  tes mct.o lecture.o dijkstra.o
  ./tes Carrefour_Pleyel  Gare_de_l_Est

mct.o:  mct.c mct.h constantes.h  lecture.h structures.h
    dijkstra.c dijkstra.h
  gcc $(CFLAGS) mct.c

lecture.o:  mct.c mct.h constantes.h  lecture.h
    structures.h dijkstra.c dijkstra.h
    gcc $(CFLAGS) lecture.c

dijkstra.o: mct.c mct.h constantes.h lecture.h structures.h
    dijkstra.c dijkstra.h
    gcc $(CFLAGS) dijkstra.c

clean:
  rm  -f  mct
  rm  -f  tes
  rm  -f  mct.o
  rm  -f  lecture.o
  rm  -f  dijkstra.o
```

../src/Makefile